

`postdetector`

September 10, 2019
13:06

Contents

1	Compute detector signals in a post-processing mode	1
2	Compute signal	15
3	INDEX	27

1 Compute detector signals in a post-processing mode

This code uses the *gpi_views* subroutine, e.g., as in the example *gpicamera.web* file to compute the end points of the chord representing each camera pixel as well as the contributions of each zone to that chord's signal. Since the *gpi_views* subroutine depends on data specific to the GPI application, it cannot be easily adapted to other situations. This version has been generalized in that it handles both D₂ (with photons coming from D, D₂, and D₂⁺) and He puffed gases and has been adapted to multiple GPI applications.

By default, *postdetector* runs with output summed over source groups. Output from an individual source group can be obtained by specifying its number on the command line:

```
postdetector 1
```

The only problem specific information appearing in this file are the camera resolution and the target plane. The data for the latter are hardcoded here with the specific applications identified with the *APP* macro. In principle, these could be moved to an input file. However, the camera resolution must remain in macro statements since the corresponding parameters appear in dimension statements. Note that all of the application specific information must match that in the corresponding *gpi_views* routine (see, e.g., *gpicamera.web*). Note also that *gpicamera.web* is to be used with the *APP = NSTX* option here. The other values of *APP* are paired with corresponding variants of the *detector_setup* routine in *gpicamera.web* that are *not* provided with DEGAS 2.

In addition to the total emission rate (i.e., the quantity to be compared with the experimentally observed emission), various other quantities are computed to facilitate checking the simulated camera geometry (e.g., the “target” R and Z) and to facilitate interpretation of the results (e.g., the “effective densities”). Regarding the latter, no single quantity has been identified as the most useful. For examples, see D. P. Stotler et al., Contrib. Plasma Phys. **44**, 294 (2004) and J. R. Myra et al., Phys. Plasmas **18**, 012305 (2011).

These quantities and the corresponding file (or variable) names are:

Name	Description	Symbol
emtt	Total emission rate ($\text{W}/(\text{m}^2 \text{ sr})$)	S
emsp	Emission rate by species p ($\text{W}/(\text{m}^2 \text{ sr})$)	S_p
tarr	Target major radius (m)	$R(j_i)$
tarz	Target vertical coordinate (m)	$Z(j_i)$
tate	Target electron temperature (eV)	$T_e(j_i)$
tane	Target electron density (m^{-3})	$n_e(j_i)$
fwpa	Emission rate per atom (W)	$F(j_i) = S(j_i)/n_1(j_i)$
nef1	Effective test density 1 ($\text{m}^{-2} \text{ sr}^{-1}$)	$S/F(j_i)$
favg	Average emission rate per atom (W)	$\langle F \rangle \equiv \langle S \rangle / \langle n_1 \rangle$
nef3	Effective test density 3 ($\text{m}^{-2} \text{ sr}^{-1}$)	$S/\langle F \rangle$
fwps	Specific emission rate per atom (W)	$S_1/n_1(j_i)$
nsf1	Specific effective test density 1 ($\text{m}^{-2} \text{ sr}^{-1}$)	$S_1/F(j_i)$
nsf3	Specific effective test density 3 ($\text{m}^{-2} \text{ sr}^{-1}$)	$S_1/\langle F_1 \rangle$
tnsp	Target density of species p (m^{-3})	$n_p(j_i)$
n2sp	Effective density 2 of species p ($\text{m}^{-2} \text{ sr}^{-1}$)	$n_{\text{eff2,p}}$
fasp	Average specific emission rate per species p (W)	$\langle F_p \rangle \equiv \langle S_p \rangle / \langle n_p \rangle$

Where:

- “Species p” refers to the p^{th} species in the problem species list.
- If the “emission rate by species” tally is present, the emission rate and derived quantities can be divided into contributions from the emitting species (e.g., D, D_2^+ , D_2). One of these is designated as the “principal” emitter (“ $p=1$ ” e.g., D).

- The function F is the “atomic physics” function for the principal emitter, i.e., the emission rate per atom evaluated at the local electron density and temperature from the input atomic physics tables (i.e., the density of the excited state relative to ground multiplied by the Einstein coefficient and energy of the transition).
- The “target” quantities are set first by identifying the zone (j_i) along each chord (i) closest to the target plane and then utilizing the corresponding quantity for that zone.
- The “average” quantities (e.g., $\langle S \rangle$) are averaged along a viewing chord over zones within a specified distance (d_{l0} in the code; e.g., 5 cm) of the target plane.
- The second effective density is the emitting particle’s density integrated along the entire viewing chord (i.e., computed just as is the total emission rate).
- These four characters in the file name will be followed by a string of the form `sg?` where ? represents the source group number, with `sg0` indicating the sum over all groups (default).

```
$Id: 834c22f8502539519a9a1ba42d26f60361e1e6a6 $  
"postdetector.f" 1 ≡  
@m FILE 'postdetector.web'  
@m MPI_messages 0  
@m CHORD_DATA 0  
@m chord_stride 5
```

The main program.

```
"postdetector.f" 1.1 ≡
program postdetector
  implicit none f77
  mp_common
  so_common
  implicit none f90

  integer nargs, o_grp
  character*LINELEN arg

  mp_decls
  sy_decls
  st_decls

  mpi_init

  if (mpi_master)
    call readfilenames
    call degas_init // In degasinit.web

  if (mpi_master) then /* This is not MPI-aware. Instead, just need to extract the required data
                        and broadcast it to the slaves. */
    call nc_read_output // In pmimatread.web

    nargs = arg_count()
    if (nargs > 0) then
      assert(nargs ≡ 1)
      call command_arg(nargs, arg)
      o_grp = read_integer(arg)
      assert(so_check(o_grp))
    else
      o_grp = 0
    end if
    call fill_views_master(o_grp)
  else
    call fill_views_slave
  end if

  mpi_end

  stop
end
```

⟨ Functions and Subroutines 1.2 ⟩

The master routine for evaluating the views.

```
"postdetector.f" 1.2 ≡
@m NSTX 0 // Macro for application. Could specify different
@m CMOD_XPT 1 // nx and ny here if needed.
@m CMOD_MID 2
@m NSTX_2010 3
@m NSTX_ENDD 4

@m APP NSTX

@if 0
@if (APP ≡ NSTX_2010)
@m nx 64
@m ny 80
#elif (APP ≡ NSTX_ENDD)
@m nx 123 // Note that the actual camera pixel indices
@m ny 56 // do not begin at 1.
#else // C-Mod or old NSTX
@m nx 64
@m ny 64
#endif
#ifndef
@m nx 5
@m ny 5
#endif
@m nsd 3 // Maximum number of test species yielding photons

/* Remove characters causing file name problems */
@m name_clean(s, sp) sp = s
ind_tmp = index(sp, '(')
if (ind_tmp > 0)
    sp(ind_tmp : ind_tmp) = '_'
ind_tmp = index(sp, ')')
if (ind_tmp > 0)
    sp(ind_tmp : ind_tmp) = '_'
ind_tmp = index(sp, '|')
if (ind_tmp > 0)
    sp(ind_tmp : ind_tmp) = '_'
ind_tmp = index(sp, '+')
if (ind_tmp > 0)
    sp(ind_tmp : ind_tmp) = 'p'
```

\langle Functions and Subroutines 1.2 $\rangle \equiv$

```
subroutine fill_views_master(o_grp)

define_dimen(species_ind, nsd)

define_varp(em_data, FLOAT, zone_ind)
define_varp(em_sp_data, FLOAT, species_ind, zone_ind)
define_varp(test_density, FLOAT, species_ind, zone_ind)
define_varp(zone_frags, FLOAT, zone_ind)

implicit none_f77
sp_common
pr_common
```

```

tl_common
ou_common
bk_common
zn_common
de_common
mp_common
@#if  $\neg$ HDF4
    po_common
    rf_common
@#endif
implicit_none_f90

integer o_grp // Input

integer zone, test, ix, iy, inverted_iy, algorithm, nviews, nfrag, nparcel, nslaves, j, nstart, num,
slave, send_data, iview, close_zone, nx_peak, iy_peak, is, total_tally, species_tally, itally, len,
ns, ind_tmp, igrp, inum, ifrag, this_grp, view_ptr
integer index_parameters tl_index_max, target_map nx,ny, target_map_slave nx,ny
real em_temp, em_rate, halfwidth, signal, a0_p, a1_p, a2_p, dist, min_dist, opt, max_opt, em_avg,
a0, a1, a2, a3, a3_p, phi, d_l0, d_opt, peak, cyl, cy_r
real x_array nx, y_array ny, signal_array nx,ny, signal_array_slave nx,ny, temp_array nx,ny,
target_r nx,ny, target_z nx,ny, target_n nx,ny,nsd, w_per_atom nx,ny, w_sp_per_atom nx,ny,
n_eff1 nx,ny, n_sp_eff1 nx,ny, n_eff2 nx,ny,nsd, n_eff2_slave nx,ny,nsd, fwpa_avg nx,ny,
fwpa_avg_slave nx,ny, n_eff3 nx,ny, n_sp_eff3 nx,ny, target_te nx,ny, target_ne nx,ny,
signal_sp_array nx,ny,nsd, fwpa_sp_avg nx,ny,nsd, signal_sp_array_slave nx,ny,nsd,
fwpa_sp_avg_slave nx,ny,nsd, profile ny, n_eff2_sum nsd, signal_sp nsd, n_avg nsd, em_sp_avg nsd
character*sp_sy_len clean_sy
character*1 s
character*3 s_grp
character*10 name_grp
logical have_zone_frags
@#if CHORD_DATA
    real r_zone, phi_zone, chord_length
    character*2 ixl, iylab
        vc_decl(chord_vec)
@#endif
@#if  $\neg$ HDF4
    integer e, fileid
    character*FILELEN postdet_file
        po_ncdecl
        nc_decls
@#endif
vc_decl(points de_view_start:de_view_end)
external extract_output_datum, gpi_views
real extract_output_datum

declare_varp(em_data)
declare_varp(em_sp_data)
declare_varp(test_density)
declare_varp(zone_frags)

{ Memory allocation interface 0 }
st_decls

```

```

vc_decls
mp_decls
zn_decls
tl_decls

var_alloc(em_data)
var_alloc(em_sp_data)
var_alloc(test_density)
var_alloc(zone_frags)

do zone = 1, zn_num
    em_datazone = zero
    do is = 1, nsd
        em_sp_datazone,is = zero
        test_densityzone,is = zero
    end do
end do

@if ~HDF4 // Use netCDF format

po_nx = nx
po_ny = ny
po_num_dep_vars = 0
var_realloc(po_xscale)
var_realloc(po_yscale)
var_realloc(po_image_data)
var_realloc(po_image_labels)
var_realloc(po_image_units)
var_realloc(po_image_formats)

#endif
total_tally = int_undef
species_tally = int_undef
do itally = 1, tl_num
    len = string_length(tally_nameitally)
    if (len > 12) then
        if (tally_nameitally SP(len - 12 :) ≡ 'emission_rate')
            total_tally = itally
    end if
    if (len > 23) then
        if (tally_nameitally SP(len - 23 :) ≡ 'emission_rate_by_species')
            species_tally = itally
    end if
end do
assert(tl_check(total_tally))
if (tl_check(species_tally)) then
    ns = pr_test_num - 1 // Skip the "geometry" test species
else
    ns = 1 // Only interested in one species; assume it's the first.
end if

em_temp = zero
do zone = 1, zn_num
    if (zn_type(zone) ≡ zn_plasma) then
        index_parameterstl_index_zone = zone
        if (o_grp ≡ 0) then

```

```

em_rate = extract_output_datum(index_parameters, 1, out_post_all, o_mean,
                               tally_name_total_tally)
s_grp = 'sg0'
name_grp = 'all\u222agroups'
else
  em_rate = extract_output_datum(index_parameters, 1, out_post_grp o_grp,0,o_mean, o_mean,
                                 tally_name_total_tally)
  write(s_grp, '(a2,i1)') 'sg', o_grp
  write(name_grp, '(a8,i1)') 'src\grp\', o_grp
end if
em_data_zone = em_rate
em_temp += em_rate * zn_volume(zone)
do test = 2, pr_test_num // Skip the "geometry" test species
  index_parameters tl_index_test = test
  is = test - 1
  if (o_grp ≡ 0) then
    test_density_zone,is = extract_output_datum(index_parameters, 1, out_post_all, o_mean,
                                                 'neutral\ndensity')
  else
    test_density_zone,is = extract_output_datum(index_parameters, 1, out_post_grp o_grp,0,o_mean,
                                                 o_mean, 'neutral\ndensity')
  end if
  if (tl.check(species_tally)) then
    if (o_grp ≡ 0) then
      em_rate = extract_output_datum(index_parameters, 1, out_post_all, o_mean,
                                      tally_name_species_tally)
    else
      em_rate = extract_output_datum(index_parameters, 1, out_post_grp o_grp,0,o_mean,
                                      o_mean, tally_name_species_tally)
    end if
    em_sp_data_zone,is = em_rate
  end if
  end do
end if
end do
@if MPI
  call MPI_bcast(em_data, zn_num, mpi_real, mpi_degas2_root, MPI_COMM_WORLD, mpi_err)
  call MPI_bcast(em_sp_data, zn_num*nsd, mpi_real, mpi_degas2_root, MPI_COMM_WORLD, mpi_err)
  call MPI_bcast(test_density, zn_num*nsd, mpi_real, mpi_degas2_root, MPI_COMM_WORLD, mpi_err)
  call MPI_bcast(ns, 1, mpi_int, mpi_degas2_root, MPI_COMM_WORLD, mpi_err)
#endifif
if (em_temp > zero) then
  nviews = nx * ny
  have_zone frags = F
  igrp = int_unused
  if (de_grps > 0) then
    do igrp = 1, de_grps
      if (detector_num_views_igrp ≡ nviews) then
        write(stdout, *) 'Using precomputed chord data for ', detector_name_igrp
        have_zone frags = T
        this_grp = igrp
      end if
    end do
  end if
endif

```

```

    end do
  end if /* Copied this logic directly from do_flights_master. */
@if MPI
  nslaves = mpi_size - 1 /* With precomputed zone_frags, should just be running in serial mode.
                           However, extending the code would only entail broadcasting the flag and the de class. */
  if (have_zone_frags) then
    assert('Not set up for MPI with precomputed zone_frags' == ' ')
  end if
#else
  nslaves = 0
#endif
  nfrag = max(nviews / max(100, 10 * nslaves), 1) // Size of each chunk
  nparcel = (nviews + nfrag - 1) / nfrag // Number of chunks

#if MPI
  do j = 0, min(nparcel, nslaves) - 1 // Send out the initial chunks
    nstart = j * nfrag
    num = min(nfrag, nviews - nstart)
    slave = j + 1
    #if MPI_messages
      write(stderr, *) 'Send_initial_batch_of ', num, ' to slave ', slave
    #endif
    call slave_send_views(slave, nstart, num)
  end do
#endif
  do j = min(nparcel, nslaves), nparcel - 1
    // Loop reading results and making additional requests
    nstart = j * nfrag
    num = min(nfrag, nviews - nstart)

#if MPI
  if (nslaves > 0) then
    call slave_receive_flag(slave)
    #if MPI_messages
      write(stderr, *) 'Slave ', slave, ' is done with a batch'
      write(stderr, *) 'Sending ', nfrag, ' more to slave ', slave
    #endif
    send_data = FALSE
    call slave_send_flag(slave, send_data)
    call slave_send_views(slave, nstart, num)
  else
    assert('Why is nslaves = 0 in an MPI run?' == ' ')
  end if
#else
  < Compute Signal 2>
#endif
  end do

#if MPI
  do j = 0, min(nparcel, nslaves) - 1 // Retrieve data from slaves this time
    call slave_receive_flag(slave)
    send_data = TRUE
    call slave_send_flag(slave, send_data)

```

```

call slave_receive_data(slave, signal_array_slave, target_map_slave, fwpa_avg_slave,
    n_eff2_slave, signal_sp_array_slave, fwpa_sp_avg_slave)

do iy = 1, ny
    do ix = 1, nx
        if (signal_array_slaveix,iy ≠ real_unused) then
            signal_arrayix,iy = signal_array_slaveix,iy
            assert(target_map_slaveix,iy ≠ int_unused)
            target_mapix,iy = target_map_slaveix,iy
            fwpa_avgix,iy = fwpa_avg_slaveix,iy
            do is = 1, ns
                assert(n_eff2_slaveix,iy,is ≠ real_unused)
                n_eff2ix,iy,is = n_eff2_slaveix,iy,is
                signal_sp_arrayix,iy,is = signal_sp_array_slaveix,iy,is
                fwpa_sp_avgix,iy,is = fwpa_sp_avg_slaveix,iy,is
            end do
        end if
    end do
end do
end do

do j = 0, nslaves - 1 // Kill off slaves
    slave = j + 1
    call slave_send_views(slave, 0, 0)
end do

@#endif

do ix = 1, nx
    x_arrayix = areal(ix - 1)
end do

do iy = 1, ny
@#if (APP ≡ NSTX_2010 ∨ APP ≡ CMOD_MID)
    y_arrayiy = areal(iy - 1)
@#else
    inverted_iy = ny - iy + 1
    y_arrayinverted_iy = areal(iy - 1)
@#endif
end do

call wrdatap(signal_array, x_array, y_array, 1, nx, 1, ny,
    'total_emission_rate' || name_grp, 'W/(m²sr)', 'emtt' || s_grp, 'E11.3')

if (ns > 1) then
    do is = 1, ns
        do iy = 1, ny
            do ix = 1, nx
                temp_arrayix,iy = signal_sp_arrayix,iy,is
            end do
        end do
    end do
    name_clean(sp_sy(pr_test(is + 1)), clean_sy) // see geomtesta.web
    assert(ns < 10)
    write(s, '(i1)') is
    call wrdatap(temp_array, x_array, y_array, 1, nx, 1, ny,
        'emission_rate_by' || trim(clean_sy) || ' ' || name_grp, 'W/(m²sr)',
        'ems' || s || s_grp, 'E11.3')

```

```

    end do
end if

@if 0
  nx_peak = 33 // I.e., at cx = 32
  iy_peak = 0
  peak = zero
  do iy = 1, ny
    profile_iy = signal_array_nx_peak,iy
    if (profile_iy > peak) then
      peak = profile_iy
      iy_peak = iy
    end if
  end do
  assert(peak > zero)
  assert(iy_peak > 0)

  assert(half * peak > profile_1)
  cy_l = zero
  cy_r = zero
  do iy = 1, ny
    if ((cy_l == zero) ∧ (profile_iy > half * peak)) then
      cy_l = y_array_iy-1 - (half * peak - profile_iy-1) / (profile_iy - profile_iy-1)
    end if
    if ((cy_r == 0) ∧ (cy_l > zero) ∧ (profile_iy < half * peak)) then
      cy_r = y_array_iy-1 - (half * peak - profile_iy-1) / (profile_iy - profile_iy-1)
    end if
  end do
  open(unit = diskout, file = 'detectorprofile.txt', status = 'unknown')
  write(diskout, *) 'At cx = ', x_array_nx_peak, ', peak = ', at cy = ', y_array_iy_peak
  write(diskout, *) 'Half peak at cy = ', cy_l, ', and ', cy_r, ', FWHM = ', cy_r - cy_l
  write(diskout, *) 'iy profile'
  do iy = 1, ny
    write(diskout, *) y_array_iy, profile_iy
  end do
  close(unit = diskout)
#endif
do iy = 1, ny
  do ix = 1, nx
    if (zn_check(target_map_ix,iy)) then
      zone = target_map_ix,iy
      target_r_ix,iy = sqrt(zone_center2_zone,1 + zone_center2_zone,2)
      target_z_ix,iy = zone_center_zone,3
      target_te_ix,iy = bk_temp(1, zone) / electron_charge
      target_ne_ix,iy = bk_n(1, zone)
      do is = 1, ns
        target_n_ix,iy,is = test_density_zone,is
      end do
      if (test_density_zone,1 > zero) then
        w_per_atom_ix,iy = em_data_zone / test_density_zone,1 /* This is just the emission by the
          principal species in the multi-species case. Skip the other permutations. */
        w_sp_per_atom_ix,iy = em_sp_data_zone,1 / test_density_zone,1
      else

```

```

w_per_atomix, iy = zero
w_sp_per_atomix, iy = zero
end if
else
  target_rix, iy = zero
  target_zix, iy = zero
  target_teix, iy = zero
  target_neix, iy = zero
  do is = 1, ns
    target_nix, iy, is = zero
  end do
  w_per_atomix, iy = zero
  w_sp_per_atomix, iy = zero
end if
if (w_per_atomix, iy > zero) then
  n_eff1ix, iy = signal_arrayix, iy / w_per_atomix, iy
else
  n_eff1ix, iy = zero
end if
if (w_sp_per_atomix, iy > zero) then
  n_sp_eff1ix, iy = signal_sp_arrayix, iy, 1 / w_sp_per_atomix, iy
else
  n_sp_eff1ix, iy = zero
end if
if (fwpa_avgix, iy > zero) then
  n_eff3ix, iy = signal_arrayix, iy / fwpa_avgix, iy
end if
if (fwpa_sp_avgix, iy, 1 > zero) then
  n_sp_eff3ix, iy = signal_sp_arrayix, iy, 1 / fwpa_sp_avgix, iy, 1
else
  n_sp_eff3ix, iy = zero
end if
end do
end do
end if

call wrdatap(target_r, x_array, y_array, 1, nx, 1, ny, 'target_major_radius', 'm',
             'tarr' || s_grp, 'E11.3')
call wrdatap(target_z, x_array, y_array, 1, nx, 1, ny, 'target_vertical_coord', 'm',
             'tarz' || s_grp, 'E11.3')
call wrdatap(target_te, x_array, y_array, 1, nx, 1, ny, 'target_electron_temperatu\
re', 'eV', 'tate' || s_grp, 'E11.3')
call wrdatap(target_ne, x_array, y_array, 1, nx, 1, ny, 'target_electron_density', 'm^3',
             'tane' || s_grp, 'E11.3')
call wrdatap(w_per_atom, x_array, y_array, 1, nx, 1, ny,
             'emission_rate_per_atom' || name_grp, 'W', 'fwpa' || s_grp, 'E11.3')
call wrdatap(n_eff1, x_array, y_array, 1, nx, 1, ny, 'effective_test_density_1\ \
' || name_grp, 'm^2_sr^-1', 'nef1' || s_grp, 'E11.3')
call wrdatap(fwpa_avg, x_array, y_array, 1, nx, 1, ny, 'avg_emission_rate_per_ato\
m' || name_grp, 'W', 'favg' || s_grp, 'E11.3')
call wrdatap(n_eff3, x_array, y_array, 1, nx, 1, ny, 'effective_test_density_3\ \
' || name_grp, 'm^2_sr^-1', 'nef3' || s_grp, 'E11.3')

```

```

if (tl_check(species_tally)) then
  call wrdatap(w_sp_per_atom, x_array, y_array, 1, nx, 1, ny, 'specific_emission_rate_pe' \\
    r_atom' || name_grp, 'W', 'fwps' || s_grp, 'E11.3')
  call wrdatap(n_sp_eff1, x_array, y_array, 1, nx, 1, ny, 'specific_effective_test_d' \\
    ensity_1' || name_grp, 'm^2_sr^-1', 'nsf1' || s_grp, 'E11.3')
  call wrdatap(n_sp_eff3, x_array, y_array, 1, nx, 1, ny, 'specific_effective_test_d' \\
    ensity_3' || name_grp, 'm^2_sr^-1', 'nsf3' || s_grp, 'E11.3')
end if

do is = 1, ns
  name_clean(sp_sy(pr_test(is + 1)), clean_sy)
  write(s, '(i1)') is
  do iy = 1, ny
    do ix = 1, nx
      temp_arrayix,iy = target_nix,iy,is
    end do
  end do
  call wrdatap(temp_array, x_array, y_array, 1, nx, 1, ny,
    'target_density_of' || trim(clean_sy) || ' ' || name_grp, 'm^3', 'tns' || s || s_grp,
    'E11.3')
  do iy = 1, ny
    do ix = 1, nx
      temp_arrayix,iy = n_eff2ix,iy,is
    end do
  end do
  call wrdatap(temp_array, x_array, y_array, 1, nx, 1, ny,
    'effective_density_2_of' || trim(clean_sy) || ' ' || name_grp, 'm^2_sr^-1',
    'n2s' || s || s_grp, 'E11.3')
  do iy = 1, ny
    do ix = 1, nx
      temp_arrayix,iy = fwp_sp_avgix,iy,is
    end do
  end do
  call wrdatap(temp_array, x_array, y_array, 1, nx, 1, ny, 'avg_specific_emission_rat' \\
    'e_per' || trim(clean_sy) || ' ' || name_grp, 'W', 'fas' || s || s_grp, 'E11.3')
end do

var_free(em_data)
var_free(em_sp_data)
var_free(test_density)
var_free(zone_frags)
@#if ~HDF4
  var_reallocb(po_xscale)
  var_reallocb(po_yscale)
  var_reallocb(po_image_data)
  var_reallocb(po_image_labels)
  var_reallocb(po_image_units)
  var_reallocb(po_image_formats)
  e = index(filenames_array_outputfile, '.nc') - 1
  assert(e > 0)
  postdet_file = filenames_array_outputfile SP(:e) || '_post.nc'
  fileid = nccreate(postdet_file, NC_CLOBBER, nc_stat)
  po_ncdef(fileid)

```

```
call ncedef(fileid, nc_stat)
po_ncwrite(fileid)
call ncclose(fileid, nc_stat)
var_free(po_xscale)
var_free(po_yscale)
var_free(po_image_data)
var_free(po_image_labels)
var_free(po_image_units)
var_free(po_image_formats)
@#endif
return
end
```

See also sections 2.1, 2.2, 2.3, 2.4, 2.5, 2.6, and 2.7.

This code is used in section 1.1.

2 Compute signal

This code is repeated in the master and slave *fill_views* routines.

```

⟨ Compute Signal 2 ⟩ ≡
    /* Specific parameters for this application */
@#if (APP ≡ NSTX)
    a0 = -const(3.60752)    // Coefficients of the camera's "target plane"
    a1 = const(2.11865)
    a2 = const(3.17544)
    a3 = -one
    phi = const(6.146, 1) * PI / const(1.8, 2)    // Toroidal shift to my system
    d_l0 = const(5., -2)    // Range for averaging about target plane
    d_opt = const(2.5, -1)    // Range for searching for a close zone
@#elif (APP ≡ CMOD_MID)
    a0 = -sin(const(-18.48) * PI / const(1.8, 2))    // A vertical plane at this
    a1 = cos(const(-18.48) * PI / const(1.8, 2))    // toroidal angle
    a2 = zero
    a3 = zero
    phi = zero    // No additional toroidal shift needed
    d_l0 = const(5., -2)
    d_opt = const(3.0, -1)
@#elif (APP ≡ CMOD_XPT)
    a0 = -sin(const(33.0) * PI / const(1.8, 2))    // Again, a vertical plane
    a1 = cos(const(33.0) * PI / const(1.8, 2))
    a2 = zero
    a3 = zero
    phi = zero
    d_l0 = const(5., -2)
    d_opt = const(3.0, -1)
@#elif (APP ≡ NSTX_2010)
    a0 = -const(3.58653)    // Coefficients of the camera's "target plane"
    a1 = const(2.08796)
    a2 = const(3.18466)
    a3 = -one
    phi = const(6.146, 1) * PI / const(1.8, 2)    // Toroidal shift to my system
    d_l0 = const(5., -2)    // Range for averaging about target plane
    d_opt = const(2.5, -1)    // Range for searching for a close zone
@#elif (APP ≡ NSTX_ENDD)
    a0 = -sin(const(155.0) * PI / const(1.8, 2))    // Again, a vertical plane
    a1 = cos(const(155.0) * PI / const(1.8, 2))
    a2 = zero
    a3 = zero
    phi = zero
    d_l0 = const(5., -2)
    d_opt = const(3.0, -1)
@#else
    assert('Target_plane_parameters_not_specified' ≡ ' ')
@#endif
    /* Rotate Ricky's "target plane" to my coordinate system. The "p" denotes "prime". */
    a0_p = a0 * cos(phi) + a1 * sin(phi)
    a1_p = a1 * cos(phi) - a0 * sin(phi)

```

```

 $a2_p = a2$ 
 $a3_p = a3$ 
do iview = nstart, nstart + num - 1
  if (mod(iview + 1, nx) ≠ 0) then
    iy = (iview + 1) / nx + 1
    ix = (iview + 1) - (iy - 1) * nx
  else
    iy = (iview + 1) / nx
    ix = nx
  end if
  if (have_zone frags) then
    assert((this_grp ≥ 1) ∧ (this_grp ≤ de_grps))
    do zone = 1, zn_num
      zone frags zone = zero
    end do
    view_ptr = de_view_pointer(iview + 1, this_grp)
    do inum = 1, de_zone frags num view_ptr
      ifrag = de_zone frags start view_ptr + (inum - 1)
      zone = de_zone frags zones ifrag
      assert(zn_check(zone))
      zone frags zone = de_zone frags ifrag
    end do
  else
    call gpi_views(ix, nx, iy, ny, vc_args(points de_view_start), halfwidth, algorithm, zone frags)
  end if
@#if CHORD_DATA
  if ((mod(ix, chord_stride) ≡ 0) ∧ (mod(iy, chord_stride) ≡ 0)) then
    write(ixlab, '(i2.2') ix
    write(iylab, '(i2.2') iy
    open(unit = diskout, file = 'gpi_chord_' || ixlab || '_' || iylab, status = 'unknown')
    write(diskout, *) 'zone zone frag' || distance || R || phi || Z || em_data || densities,
  end if
@#endif
  signal = zero
  em_avg = zero
  do is = 1, ns
    n_eff2_sum_is = zero
    signal_sp_is = zero
    n_avg_is = zero
    em_sp_avg_is = zero
  end do
  min_dist = const(1., 16)
  max_opt = const(1., -16)
  close_zone = int_undef
  do zone = 1, zn_num
@#if CHORD_DATA
  if ((zone frags zone > zero) ∧ (mod(ix, chord_stride) ≡ 0) ∧ (mod(iy, chord_stride) ≡ 0)) then
    r_zone = sqrt(zone_center2zone,1 + zone_center2zone,2)
    phi_zone = atan2(zone_centerzone,2, zone_centerzone,1)
    vc_difference(zone_centerzone, points de_view_start, chord_vec)
    chord_length = vc_abs(chord_vec)
    write(diskout, '(2x,i7,2x,1p,e16.8,0p,4(2x,f11.8),1p,4(2x,e16.8))' zone,

```

```

    zone_frags_zone * zn_volume(zone), chord_length, r_zone, phi_zone, zone_center_zone,3,
    em_data_zone, (test_density_zone,is, is = 1, ns)
end if
@#endif
signal += em_data_zone * zone_frags_zone * zn_volume(zone)
do is = 1, ns
    n_eff2_sum_is += test_density_zone,is * zone_frags_zone * zn_volume(zone)
    signal_sp_is += em_sp_data_zone,is * zone_frags_zone * zn_volume(zone)
end do /* Hack to distill out zone closest to target plane. */
dist = abs(a0_p * zone_center_zone,1 + a1_p * zone_center_zone,2 + a2_p * zone_center_zone,3 + a3_p) /
    sqrt(a0_p^2 + a1_p^2 + a2_p^2)
if (dist ≠ zero) then
    opt = zone_frags_zone * zn_volume(zone) / dist
else /* In the C-Mod problems, can actually get dist = 0. Use a finite length for computing
       opt to distinguish these in-target-plane zones. */
    opt = zone_frags_zone * zn_volume(zone) / const(1., -8)
end if
@if 0
if ((zone_frags_zone > zero) ∧ (dist < min_dist)) then
@#endif
if ((opt > max_opt) ∧ (dist ≤ d_opt)) then
    max_opt = opt
    min_dist = dist
    close_zone = zone
end if /* Sums restricted to region around target plane. */
if (dist ≤ d_l0) then
    em_avg += em_data_zone * zone_frags_zone * zn_volume(zone)
    do is = 1, ns
        n_avg_is += test_density_zone,is * zone_frags_zone * zn_volume(zone)
        em_sp_avg_is += em_sp_data_zone,is * zone_frags_zone * zn_volume(zone)
    end do
end if
end do
@if CHORD_DATA
if ((mod(ix, chord_stride) ≡ 0) ∧ (mod(iy, chord_stride) ≡ 0))
    close(unit = diskout)
@#endif
@if (APP ≡ NSTX_2010 ∨ APP ≡ NSTX_ENDD ∨ CMOD_MID)
    inverted_iy = iy
@#else
    inverted_iy = ny - iy + 1
@#endif
/* Have a case now in which a chord does not reach the target plane. Will let close_zone remain
   int_undef and then assign zero values to the associated target parameters. */
@if 0
assert(close_zone ≠ int_undef)
assert(min_dist < const(1., 16))
@#endif
target_map_ix,inverted_iy = close_zone
signal_array_ix,inverted_iy = signal
if (n_avg_1 > zero) then
    fwpa_avg_ix,inverted_iy = em_avg / n_avg_1

```

```
else
    fwpa_avgix,inverted_iy = zero
end if
do is = 1, ns
    n_eff2ix,inverted_iy,is = n_eff2_sumis
    signal_sp_arrayix,inverted_iy,is = signal_spis
    if (n_avgis > zero) then
        fwpa_sp_avgix,inverted_iy,is = em_sp_avgis / n_avgis
    else
        fwpa_sp_avgix,inverted_iy,is = zero
    end if
end do
end do
```

This code is used in sections 1.2 and 2.1.

Slave version for filling views.

```

⟨ Functions and Subroutines 1.2 ⟩ +≡
@if MPI
  subroutine fill_views_slave

    define_varp(em_data, FLOAT, zone_ind)
    define_varp(em_sp_data, FLOAT, species_ind, zone_ind)
    define_varp(test_density, FLOAT, species.ind, zone.ind)
    define_varp(zone_frags, FLOAT, zone.ind)

    implicit none f77
    mp_common
    zn_common
    de_common
    implicit none f90

    integer iy, ix, tag, num, nstart, iview, is, ns, algorithm, close_zone, zone, inverted_iy, done_flag,
           send_data, inum, ifrag
    integer target_map_nx, ny
    real halfwidth, signal, a0_p, a1_p, a2_p, dist, min_dist, a0, a1, a2, a3, phi, d_l0, d_opt, a3_p,
           em_avg, opt, max_opt
    real signal_array_nx, ny, n_eff2_nx, ny, nsd, fwpa_avg_nx, ny, signal_sp_array_nx, ny, nsd,
           fwpa_sp_avg_nx, ny, nsd, n_eff2_sum_nsd, signal_sp_nsd, n_avg_nsd, em_sp_avg_nsd
    logical have_zone_frags
@if CHORD_DATA
    real r_zone, phi_zone, chord_length
    character*2 ixlab, iylab

    vc_decl(chord_vec)
#endif
    vc_decl(points_de_view_start:de_view_end)
    declare_varp(em_data)
    declare_varp(em_sp_data)
    declare_varp(test_density)
    declare_varp(zone_frags)

    ⟨ Memory allocation interface 0 ⟩
    vc_decls
    mp_decls
    zn_decls

    var_alloc(em_data)
    var_alloc(em_sp_data)
    var_alloc(test_density)
    var_alloc(zone_frags)

    call MPI_bcast(em_data, zn_num, mpi_real, mpi_degas2_root, MPI_COMM_WORLD, mpi_err)
    call MPI_bcast(em_sp_data, zn_num*nsd, mpi_real, mpi_degas2_root, MPI_COMM_WORLD, mpi_err)
    call MPI_bcast(test_density, zn_num*nsd, mpi_real, mpi_degas2_root, MPI_COMM_WORLD, mpi_err)
    call MPI_bcast(ns, 1, mpi_int, mpi_degas2_root, MPI_COMM_WORLD, mpi_err)

    do iy = 1, ny
      do ix = 1, nx
        signal_array_ix, iy = real_unused
        target_map_ix, iy = int_unused

```

```

 $f_{wpa\_avg}_{ix,iy} = real\_unused$ 
do  $is = 1, ns$ 
     $n_{eff2}_{ix,iy,is} = real\_unused$ 
     $signal_{-sp\_array}_{ix,iy,is} = real\_unused$ 
     $f_{wpa\_sp\_avg}_{ix,iy,is} = real\_unused$ 
end do
end do
end do

loop: continue
tag = 100
call MPI_recv( $num, 1, mpi\_int, mpi\_degas2\_root, tag, MPI\_COMM\_WORLD, mpi\_status, mpi\_err$ )
if ( $num \equiv 0$ )
    goto break
tag++
call MPI_recv( $nstart, 1, mpi\_int, mpi\_degas2\_root, tag, MPI\_COMM\_WORLD, mpi\_status, mpi\_err$ )
    /* See comment in master routine. */
have_zone_frags =  $\mathcal{F}$ 
⟨Compute Signal 2⟩

done_flag = TRUE
tag = 200
call MPI_send( $done\_flag, 1, mpi\_int, mpi\_degas2\_root, tag, MPI\_COMM\_WORLD, mpi\_err$ )
tag = 300
call MPI_recv( $send\_data, 1, mpi\_int, mpi\_degas2\_root, tag, MPI\_COMM\_WORLD, mpi\_status, mpi\_err$ )
if ( $send\_data \equiv FALSE$ ) then
    goto loop
else if ( $send\_data \equiv TRUE$ ) then
    tag = 400
    call MPI_send( $signal\_array, nx * ny, mpi\_real, mpi\_degas2\_root, tag, MPI\_COMM\_WORLD, mpi\_err$ )
    tag = 401
    call MPI_send( $target\_map, nx * ny, mpi\_int, mpi\_degas2\_root, tag, MPI\_COMM\_WORLD, mpi\_err$ )
    tag = 402
    call MPI_send( $f_{wpa\_avg}, nx * ny, mpi\_real, mpi\_degas2\_root, tag, MPI\_COMM\_WORLD, mpi\_err$ )
    tag = 403
    call MPI_send( $n_{eff2}, nx * ny * nsd, mpi\_real, mpi\_degas2\_root, tag, MPI\_COMM\_WORLD, mpi\_err$ )
    tag = 404
    call MPI_send( $signal_{-sp\_array}, nx * ny * nsd, mpi\_real, mpi\_degas2\_root, tag, MPI\_COMM\_WORLD, mpi\_err$ )
    tag = 405
    call MPI_send( $f_{wpa\_sp\_avg}, nx * ny * nsd, mpi\_real, mpi\_degas2\_root, tag, MPI\_COMM\_WORLD, mpi\_err$ )
do  $iy = 1, ny$ 
    do  $ix = 1, nx$ 
         $signal_{-array}_{ix,iy} = real\_unused$ 
         $target_{-map}_{ix,iy} = int\_unused$ 
         $f_{wpa\_avg}_{ix,iy} = real\_unused$ 
        do  $is = 1, ns$ 
             $n_{eff2}_{ix,iy,is} = real\_unused$ 
             $signal_{-sp\_array}_{ix,iy,is} = real\_unused$ 

```

```

    fwpaspa_avgix,iy,is = real_unused
end do
end do
end do
goto loop
else
assert('Illegal value of send_data' ≡ ' ')
end if

break: continue
var_free(em_data)
var_free(em_sp_data)
var_free(test_density)
var_free(zone_frags)
return
end
@#else
subroutine fill_views_slave // dummy
return
end
@#endiff

```

Send data to slave.

⟨ Functions and Subroutines 1.2 ⟩ +≡
@#if MPI

```

subroutine slave_send_views(slave, nstart, num)
implicit none_f77
mp_common
implicit none_f90
mp_decls
integer slave, nstart, num
integer tag

⟨ Memory allocation interface 0 ⟩

tag = 100
call MPI_send(num, 1, mpi_int, slave, tag, MPI_COMM_WORLD, mpi_err)
if (num ≡ 0)
    goto break
tag++
call MPI_send(nstart, 1, mpi_int, slave, tag, MPI_COMM_WORLD, mpi_err)

break: continue
return
end
@#endiff

```

Receive notice from a slave indicating that it is done with current batch.

```
⟨ Functions and Subroutines 1.2 ⟩ +≡
@if MPI
  subroutine slave_receive_flag(slave)
    implicit none_f77
    mp_common
    implicit none_f90
    mp_decls
    integer slave
    integer tag, done_flag

    ⟨ Memory allocation interface 0 ⟩

    tag = 200
    call MPI_recv(done_flag, 1, mpi_int, MPI_ANY_SOURCE, tag, MPI_COMM_WORLD, mpi_status,
                 mpi_err)
    slave = mpi_status(MPI_SOURCE)
    assert(done_flag ≡ TRUE)
    return
  end
#endif
```

Send a flag to a slave instructing it to send back data from its flights.

```
⟨ Functions and Subroutines 1.2 ⟩ +≡
@if MPI
  subroutine slave_send_flag(slave, send_data)
    implicit none_f77
    mp_common
    implicit none_f90
    mp_decls
    integer slave, send_data    // Input
    integer tag     // Local

    ⟨ Memory allocation interface 0 ⟩

    tag = 300
    call MPI_send(send_data, 1, mpi_int, slave, tag, MPI_COMM_WORLD, mpi_err)

    return
  end
#endif
```

Receive data from flights run by a slave.

⟨ Functions and Subroutines 1.2 ⟩ +≡

@#if MPI

```

subroutine slave_receive_data(slave, signal_array, target_map, fwpa_avg, n_eff2, signal_sp_array,
                               fwpa_sp_avg)
implicit none_f77
mp_common
implicit none_f90
mp_decls
integer slave
integer target_map_nx,ny
real signal_array_nx,ny, fwpa_avg_nx,ny, n_eff2_nx,ny,nsd, signal_sp_array_nx,ny,nsd,
      fwpa_sp_avg_nx,ny,nsd
integer tag

⟨ Memory allocation interface 0 ⟩

tag = 400
call MPI_recv(signal_array, nx * ny, mpi_real, MPI_ANY_SOURCE, tag, MPI_COMM_WORLD,
               mpi_status, mpi_err)
tag = 401
call MPI_recv(target_map, nx * ny, mpi_int, MPI_ANY_SOURCE, tag, MPI_COMM_WORLD,
               mpi_status, mpi_err)
tag = 402
call MPI_recv(fwpa_avg, nx * ny, mpi_real, MPI_ANY_SOURCE, tag, MPI_COMM_WORLD,
               mpi_status, mpi_err)
tag = 403
call MPI_recv(n_eff2, nx * ny * nsd, mpi_real, MPI_ANY_SOURCE, tag, MPI_COMM_WORLD,
               mpi_status, mpi_err)
tag = 404
call MPI_recv(signal_sp_array, nx * ny * nsd, mpi_real, MPI_ANY_SOURCE, tag,
               MPI_COMM_WORLD, mpi_status, mpi_err)
tag = 405
call MPI_recv(fwpa_sp_avg, nx * ny * nsd, mpi_real, MPI_ANY_SOURCE, tag,
               MPI_COMM_WORLD, mpi_status, mpi_err)

return
end
@#endif

```

Rearrange 2D array prior to writing. This configuration is specific to the 2004 NSTX Gas Puff Imaging experiments.

```

⟨ Functions and Subroutines 1.2 ⟩ +≡
subroutine wrdatap(pb2dat, xdat, ydat, kxmin, kx, kymin, ky, clabel, cunits, cfilrt, cformt)
  implicit none_f77
  implicit none_f90

  integer kxmin, kx, ky, kymin // Input
  character(*) cunits, clabel, cfilrt, cformt
  real xdat_kx, ydat_ky, pb2dat_kx,ky

  integer ix, iyp, ixprime, iypprime // Local
  real xprime_ny, yprime_nx, pb2prime_ny,nx

  assert(kx ≡ nx)
  assert(ky ≡ ny)
@#if (APP ≡ NSTX)
  assert(kx ≡ ky) /* The required mapping is equivalent to rowflip(transpose(q)) in Transform.
    Namely, swap ix and iyp, and then invert the horizontal coordinate. Note that we just assume
    that the dimensions are the same in both directions. */
  do iyp = kymin, ky
    do ix = kxmin, kx
      ixprime = iyp
      iypprime = kx - ix + 1
      pb2prime_ixprime,iypprime = pb2dat_ix,iyp
    end do
  end do /* Apply equivalent transformations to the scales. */
  do ix = kxmin, kx
    xprime_ix = ydat_ix
  end do

  do iyp = kymin, ky
    ixprime = ky - iyp + 1
    yprime_iy = xdat_ixprime
  end do /* Call wrdata with the revised arrays */
  call wrdata(pb2prime, xprime, yprime, kxmin, kx, kymin, ky, clabel, cunits, cfilrt, cformt)
@#else
  call wrdata(pb2dat, xdat, ydat, kxmin, kx, kymin, ky, clabel, cunits, cfilrt, cformt)
@#endif

  return
end

```

Write 2D array to HDF file. Borrowed this from the H collisional radiative code, irls.

```

⟨ Functions and Subroutines 1.2 ⟩ +≡
  subroutine wrdata(pb2dat, xdat, ydat, kxmin, kx, kymin, ky, clabel, cunits, cfilt, cformt)
    implicit none f77
  @#if ~HDF4
    po_common
  @#endif
    implicit none f90

  integer kxmin, kx, ky, kymin      // Input
  character*(*) cunits, clabel, cfilt, cformt
  real xdat_kx, ydat_ky, pb2dat_kx, ky
  @#if HDF4
    integer icount, iret, ix, iy, irank      // Local
    integer*4 idim2

    single_precision zdatic, zmax, zmin
    single_precision zxdat_kx, zydat_ky, zdata_kx*kx

    integer dssdast, dssdims, dssdisc, dssrang, dspdata
    external dssdast, dssdims, dssdisc, dssrang, dspdata, xerrab /* Set dimension sizes */

    irank = 2
    idim1 = kx - kxmin + 1
    idim2 = ky - kymin + 1 /* Determine maximum and minimum; assign data to 1-D work array. */
    zmax = -const(1., 20)
    zmin = const(1., 20)
    icount = 0

    do iy = kymin, ky
      zydat_iy = single(ydat_iy)
      do ix = kxmin, kx
        zxdat_ix = single(xdat_ix)
        icount++
        zdatic = pb2dat_ix, iy
        zmax = max(zmax, zdatic)
        zmin = min(zmin, zdatic)
        zdata_icount = zdatic
      end do
    end do /* HDF set calls */
    iret = dssdims(irank, idim)
    assert(iret ≡ 0)
    iret = dssdisc(1, idim1, zxdat)
    assert(iret ≡ 0)
    iret = dssdisc(2, idim2, zydat)
    assert(iret ≡ 0)
    iret = dssdast(clabel, cunits, cformt, 'cartesian')
    assert(iret ≡ 0)

    iret = dssrang(zmax, zmin)
    assert(iret ≡ 0) /* Write 2-D data */
    iret = dspdata(cfilt || '.hdf', 2, idim, zdata)
    assert(iret ≡ 0)
  @#else // !HDF

```

```

integer ix, iy, jx, jy // Local
⟨Memory allocation interface 0⟩
assert(po_nx ≡ kx - kxmin + 1)
assert(po_ny ≡ ky - kymin + 1)
po_num_dep_vars++
var_realloc(po_xscale)
var_realloc(po_yscale)
var_realloc(po_image_data)
var_realloc(po_image_labels)
var_realloc(po_image_units)
var_realloc(po_image_formats)
/* Define associated scales starting at kymin? (set auxiliary info: units, format, etc.) */
po_image_labelspo_num_dep_vars = clabel
po_image_unitspo_num_dep_vars = cunits
po_image_formatspo_num_dep_vars = cformat
do iy = kymin, ky
    jy = iy - kymin + 1
    po_yscalepo_num_dep_vars,jy = areal(iy)
    do ix = kxmin, kx
        jx = ix - kxmin + 1
        po_xscalepo_num_dep_vars,jx = areal(jx)
        po_image_datapo_num_dep_vars,jy,jx = pb2datix,iy
    end do
end do
end do
@#endif
return
end

```

3 INDEX

abs: 2.
algorithm: 1.2, 2, 2.1.
APP: 1, 1.2, 2, 2.6.
areal: 1.2, 2.7.
arg: 1.1.
arg_count: 1.1.
assert: 1.1, 1.2, 2, 2.1, 2.3, 2.6, 2.7.
atan2: 2.
a0: 1.2, 2, 2.1.
a0_p: 1.2, 2, 2.1.
a1: 1.2, 2, 2.1.
a1_p: 1.2, 2, 2.1.
a2: 1.2, 2, 2.1.
a2_p: 1.2, 2, 2.1.
a3: 1.2, 2, 2.1.
a3_p: 1.2, 2, 2.1.
bk_common: 1.2.
bk_n: 1.2.
bk_temp: 1.2.
break: 2.1, 2.2.
cflrt: 2.6, 2.7.
cformat: 2.6, 2.7.
CHORD_DATA: 1, 1.2, 2, 2.1.
chord_length: 1.2, 2, 2.1.
chord_stride: 1, 2.
chord_vec: 1.2, 2, 2.1.
clabel: 2.6, 2.7.
clean_sy: 1.2.
close_zone: 1.2, 2, 2.1.
CMOD_MID: 1.2, 2.
CMOD_XPT: 1.2, 2.
command_arg: 1.1.
const: 2, 2.7.
cos: 2.
cunits: 2.6, 2.7.
cy_l: 1.2.
cy_r: 1.2.
d_l0: 1, 1.2, 2, 2.1.
d_opt: 1.2, 2, 2.1.
de: 1.2.
de_common: 1.2, 2.1.
de_grps: 1.2, 2.
de_view_end: 1.2, 2.1.
de_view_pointer: 2.
de_view_start: 1.2, 2, 2.1.
de_zone_frags: 2.
de_zone_frags_num: 2.
de_zone_frags_start: 2.
de_zone_frags_zones: 2.
declare_varp: 1.2, 2.1.
define_dimen: 1.2.
define_varp: 1.2, 2.1.
degas_init: 1.1.
degasinit: 1.1.
detector_name: 1.2.
detector_num_views: 1.2.
detector_setup: 1.
diskout: 1.2, 2.
dist: 1.2, 2, 2.1.
do_flights_master: 1.2.
done_flag: 2.1, 2.3.
dspdata: 2.7.
dssdast: 2.7.
dssdims: 2.7.
dssdisc: 2.7.
dssrang: 2.7.
e: 1.2.
electron_charge: 1.2.
em_avg: 1.2, 2, 2.1.
em_data: 1.2, 2, 2.1.
em_rate: 1.2.
em_sp_avg: 1.2, 2, 2.1.
em_sp_data: 1.2, 2, 2.1.
em_temp: 1.2.
extract_output_datum: 1.2.
FALSE: 1.2, 2.1.
file: 1.2, 2.
FILE: 1.
fileid: 1.2.
FILELEN: 1.2.
filenames_array: 1.2.
fill_views: 2.
fill_views_master: 1.1, 1.2.
fill_views_slave: 1.1, 2.1.
FLOAT: 1.2, 2.1.
fwp_a_avg: 1.2, 2, 2.1, 2.5.
fwp_a_avg_slave: 1.2.
fwp_a_sp_avg: 1.2, 2, 2.1, 2.5.
fwp_a_sp_avg_slave: 1.2.
geomtesta: 1.2.
gpi_views: 1, 1.2, 2.
gpicamera: 1.
half: 1.2.
halfwidth: 1.2, 2, 2.1.
have_zone_frags: 1.2, 2, 2.1.
HDF4: 1.2, 2.7.
icount: 2.7.
idim: 2.7.

ifrag: 1.2, 2, 2.1.
igrp: 1.2.
implicit_none_f77: 1.1, 1.2, 2.1, 2.2, 2.3, 2.4, 2.5, 2.6, 2.7.
implicit_none_f90: 1.1, 1.2, 2.1, 2.2, 2.3, 2.4, 2.5, 2.6, 2.7.
ind_tmp: 1.2.
index: 1.2.
index_parameters: 1.2.
int_undef: 1.2, 2.
int_unused: 1.2, 2.1.
inum: 1.2, 2, 2.1.
inverted_iy: 1.2, 2, 2.1.
irank: 2.7.
iret: 2.7.
is: 1.2, 2, 2.1.
itally: 1.2.
iview: 1.2, 2, 2.1.
ix: 1.2, 2, 2.1, 2.6, 2.7.
ixlab: 1.2, 2, 2.1.
ixprime: 2.6.
iy: 1.2, 2, 2.1, 2.6, 2.7.
iy_peak: 1.2.
iylab: 1.2, 2, 2.1.
iyprime: 2.6.
j: 1.2.
jx: 2.7.
jk: 2.7.
kx: 2.6, 2.7.
kxmin: 2.6, 2.7.
ky: 2.6, 2.7.
kymin: 2.6, 2.7.
len: 1.2.
LINELEN: 1.1.
loop: 2.1.
max: 1.2, 2.7.
max_opt: 1.2, 2, 2.1.
min: 1.2, 2.7.
min_dist: 1.2, 2, 2.1.
mod: 2.
mp_common: 1.1, 1.2, 2.1, 2.2, 2.3, 2.4, 2.5.
mp_decls: 1.1, 1.2, 2.1, 2.2, 2.3, 2.4, 2.5.
MPI: 1.2, 2.1, 2.2, 2.3, 2.4, 2.5.
MPI_ANY_SOURCE: 2.3, 2.5.
MPI_bcast: 1.2, 2.1.
MPI_COMM_WORLD: 1.2, 2.1, 2.2, 2.3, 2.4, 2.5.
mpi_degas2_root: 1.2, 2.1.
mpi_end: 1.1.
mpi_err: 1.2, 2.1, 2.2, 2.3, 2.4, 2.5.
mpi_init: 1.1.
mpi_int: 1.2, 2.1, 2.2, 2.3, 2.4, 2.5.
mpi_master: 1.1.
MPI_messages: 1, 1.2.
mpi_real: 1.2, 2.1, 2.5.
MPI_recv: 2.1, 2.3, 2.5.
MPI_send: 2.1, 2.2, 2.4.
mpi_size: 1.2.
MPI_SOURCE: 2.3.
mpi_status: 2.1, 2.3, 2.5.
n_avg: 1.2, 2, 2.1.
n_eff1: 1.2.
n_eff2: 1.2, 2, 2.1, 2.5.
n_eff2_slave: 1.2.
n_eff2_sum: 1.2, 2, 2.1.
n_eff3: 1.2.
n_sp_eff1: 1.2.
n_sp_eff3: 1.2.
name_clean: 1.2.
name_grp: 1.2.
nargs: 1.1.
NC_CLOBBER: 1.2.
nc_decls: 1.2.
nc_read_output: 1.1.
nc_stat: 1.2.
ncclose: 1.2.
nccreate: 1.2.
ncendef: 1.2.
nfrag: 1.2.
nparcel: 1.2.
ns: 1.2, 2, 2.1.
nsd: 1.2, 2.1, 2.5.
nslaves: 1.2.
nstart: 1.2, 2, 2.1, 2.2.
NSTX: 1, 1.2, 2, 2.6.
NSTX_ENDD: 1.2, 2.
NSTX_2010: 1.2, 2.
num: 1.2, 2, 2.1, 2.2.
nviews: 1.2.
nx: 1.2, 2, 2.1, 2.5, 2.6.
nx_peak: 1.2.
ny: 1.2, 2, 2.1, 2.5, 2.6.
o_grp: 1.1, 1.2.
o_mean: 1.2.
one: 2.
opt: 1.2, 2, 2.1.
ou_common: 1.2.
out_post_all: 1.2.
out_post_grp: 1.2.
outputfile: 1.2.
pb2dat: 2.6, 2.7.
pb2prime: 2.6.

peak: 1.2.
phi: 1.2, 2, 2.1.
phi_zone: 1.2, 2, 2.1.
PI: 2.
pmimatread: 1.1.
po_common: 1.2, 2.7.
po_image_data: 1.2, 2.7.
po_image_formats: 1.2, 2.7.
po_image_labels: 1.2, 2.7.
po_image_units: 1.2, 2.7.
po_ncdecl: 1.2.
po_ncdef: 1.2.
po_ncwrite: 1.2.
po_num_dep_vars: 1.2, 2.7.
po_nx: 1.2, 2.7.
po_ny: 1.2, 2.7.
po_xscale: 1.2, 2.7.
po_yscale: 1.2, 2.7.
points: 1.2, 2, 2.1.
postdet_file: 1.2.
postdetector: 1, 1.1.
pr_common: 1.2.
pr_test: 1.2.
pr_test_num: 1.2.
profile: 1.2.

r_zone: 1.2, 2, 2.1.
read_integer: 1.1.
readfilenames: 1.1.
real_unused: 1.2, 2.1.
rf_common: 1.2.
rowflip: 2.6.

s: 1.2.
s_grp: 1.2.
send_data: 1.2, 2.1, 2.4.
signal: 1.2, 2, 2.1.
signal_array: 1.2, 2, 2.1, 2.5.
signal_array_slave: 1.2.
signal_sp: 1.2, 2, 2.1.
signal_sp_array: 1.2, 2, 2.1, 2.5.
signal_sp_array_slave: 1.2.
sin: 2.
single: 2.7.
single_precision: 2.7.
slave: 1.2, 2.2, 2.3, 2.4, 2.5.
slave_receive_data: 1.2, 2.5.
slave_receive_flag: 1.2, 2.3.
slave_send_flag: 1.2, 2.4.
slave_send_views: 1.2, 2.2.
so_check: 1.1.
so_common: 1.1.
sp: 1.2.

SP: 1.2.
sp_common: 1.2.
sp_sy: 1.2.
sp_sy_len: 1.2.
species_ind: 1.2, 2.1.
species_tally: 1.2.
sqrt: 1.2, 2.
st_decls: 1.1, 1.2.
status: 1.2, 2.
stderr: 1.2.
stdout: 1.2.
string_length: 1.2.
sy_decls: 1.1.

tag: 2.1, 2.2, 2.3, 2.4, 2.5.
tally_name: 1.2.
target_map: 1.2, 2, 2.1, 2.5.
target_map_slave: 1.2.
target_n: 1.2.
target_ne: 1.2.
target_r: 1.2.
target_te: 1.2.
target_z: 1.2.
temp_array: 1.2.
test: 1.2.
test_density: 1.2, 2, 2.1.
this_grp: 1.2, 2.
tl_check: 1.2.
tl_common: 1.2.
tl_decls: 1.2.
tl_index_max: 1.2.
tl_index_test: 1.2.
tl_index_zone: 1.2.
tl_num: 1.2.
total_tally: 1.2.
transpose: 2.6.
trim: 1.2.
TRUE: 1.2, 2.1, 2.3.

unit: 1.2, 2.

var_alloc: 1.2, 2.1.
var_free: 1.2, 2.1.
var_realloca: 1.2, 2.7.
var_reallocb: 1.2.
vc_abs: 2.
vc_args: 2.
vc_decl: 1.2, 2.1.
vc_decls: 1.2, 2.1.
vc_difference: 2.
view_ptr: 1.2, 2.

w_per_atom: 1.2.
w_sp_per_atom: 1.2.

web: 1, 1.1, 1.2.

wrdata: 2.6, 2.7.

wrdatap: 1.2, 2.6.

x_array: 1.2.

xdat: 2.6, 2.7.

xerrab: 2.7.

xprime: 2.6.

y_array: 1.2.

ydat: 2.6, 2.7.

yprime: 2.6.

zdata: 2.7.

zdic: 2.7.

zero: 1.2, 2.

zmax: 2.7.

zmin: 2.7.

zn_check: 1.2, 2.

zn_common: 1.2, 2.1.

zn_decls: 1.2, 2.1.

zn_num: 1.2, 2, 2.1.

zn_plasma: 1.2.

zn_type: 1.2.

zn_volume: 1.2, 2.

zone: 1.2, 2, 2.1.

zone_center: 1.2, 2.

zone_frags: 1.2, 2, 2.1.

zone_ind: 1.2, 2.1.

zxdat: 2.7.

zydat: 2.7.

⟨ Compute Signal 2⟩ Used in sections 1.2 and 2.1.

⟨ Functions and Subroutines 1.2, 2.1, 2.2, 2.3, 2.4, 2.5, 2.6, 2.7⟩ Used in section 1.1.

⟨ Memory allocation interface 0⟩ Used in sections 2.7, 2.5, 2.4, 2.3, 2.2, 2.1, and 1.2.

COMMAND LINE: "fweave -f -i! -W[-ybs15000 -ykw800 -ytw40000 -j -n/
/Users/dstotler/degas2/src/postdetector.web".

WEB FILE: "/Users/dstotler/degas2/src/postdetector.web".

CHANGE FILE: (none).

GLOBAL LANGUAGE: FORTRAN.